DMSC Working Group H. Yang
Internet-Draft T. Yu
Intended status: Standards Track Q. Yao
Expires: 4 January 2026 Z. Zhang
Beijing University of Posts and Telecommunications
3 July 2025

Microservice Communication Resource Scheduling for Distributed AI Model
draft-yang-dmsc-distributed-model-04

Abstract

This document describes the architecture of microservice communication resource scheduling for distributed AI model.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 4 January 2026.

Table of Contents

1. Introduction 2
2. Conventions used in this document 3
3. Terminology 3
4. DMSC-LMT architecture 4
4.1. Overview 4
4.2. Function modules 6
4.3. The control signaling messages design of DMSC-LMT 8
4.4. DMSC-LMT communication flow 9
4.5. Task-Level Microservice Decomposition 11
5. Realization of key functions 12
5.1. Efficient Microservice Communication and Orchestration . 13
5.2. Intelligent decision support for functional modules . . . 13
5.3. Task topology and dependency control 13
5.4. Computationally aware routing and congestion avoidance mechanisms 13
5.5. Fault handling and communication resilience 14
6. Local Database Integration and State Management in DMSC-LMT 14
6.1. Roles and Functions of Local Databases 14
6.2. Coordination Across Instances 15
6.3. Integration with Control Plane Components 15
7. The specific process of DMSC-LTM 16
7.1. Overview of Large Model Training Parallel Strategies . . 16
7.2. Detailed mapping of distributed training to DMSC-LTM . . 19
8. Conclusion and Outlook 21
9. IANA Considerations 21
10. Acknowledgement 21
11. References 21
11.1. Normative References 21
11.2. Informative References 22
Appendix A. An Appendix 22
Authors' Addresses 22

1. Introduction

With the rapid advancement of Large Models such as GPT, Grok, and DeepSeek, training workloads have continuously increased in terms of scale, complexity, and resource consumption.This trend imposes stricter requirements on compute resource scheduling, communication efficiency, system resilience, and overall scalability.

Traditional centralized control and monolithic architectures face several challenges, including network congestion, load imbalances, and difficulties in failure recovery. While current mainstream training systems typically rely on multi-node distributed parallel frameworks to improve training throughput through task partitioning

and communication synchronization, they still encounter significant limitations when dealing with ultra-large model parameters, heterogeneous hardware clusters, and high-density communication patterns. These limitations include coarse scheduling granularity, inefficient communication path optimization, and inadequate fault tolerance, which ultimately lead to scalability bottlenecks and degraded system performance.

To address these challenges, this draft proposes the Distributed Microservice Communication Architecture for Large Model Training (DMSC-LMT). DMSC-LMT is designed to enhance scheduling flexibility, communication efficiency, system resilience, and scalability in high-concurrency, large-scale, and heterogeneous training environments by introducing mechanisms such as task-level microservice decomposition, content-semantic addressing, and computation-aware routing. This architecture provides a unified, extensible, and evolvable foundation for building the next generation of Large Model training platforms.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Terminology

The following terms are defined in this draft:

* DMSC-LMT: Distributed Micro Service Communication architecture, a Distributed Microservice Communication Architecture for large model training defined in this draft.

* MG: Microservice gateway, a core component in the microservice architecture that serves as a unified entry point for external requests. It is responsible for traffic routing, load balancing, service authentication, authorization, request filtering, logging, API aggregation, and other tasks to ensure that external requests are efficiently and stably dispatched to the service instance.

* SR: Service Router, responsible for internal communication between microservices, dynamically selects the appropriate service instance using the service discovery mechanism, and performs traffic routing and load distribution based on the traffic scheduling algorithm to ensure efficient and stable communication between services.

* SRD: Service Registry and Discovery, responsible for centrally managing the registration and discovery of microservices. Microservices register their instance information with the service

registry upon startup, and SRD provides service discovery and registration information to SR and MG, ensuring they can dynamically locate service instances and route traffic.

* AAM: Authentication and Authorization Module,it is the core component responsible for handling identity authentication and authorization in MG. It ensures that all incoming requests are authenticated and have appropriate permissions to protect the system from unauthorized access.

* MI:Microservice Instance,A runtime-executable entity derived from a decomposed task, identified by a unique Service ID (SID), and deployed within a Microservice Instance Cluster (MIC). It performs a specific training-stage function and participates in the service routing and discovery ecosystem.

* MIC: Microservice instance cluster,refers to a collection of microservice instances that collaborate in a distributed system to provide specific functions or services. Each MIC contains a set of microservice instances (such as a /1, B/1), and the instances in the cluster can perform the same or different tasks, serving as the basic computing unit for data parallelism, model parallelism, or hybrid training strategies. The cluster’s microservices are coordinated by SRD and SR to ensure seamless communication and data synchronization.

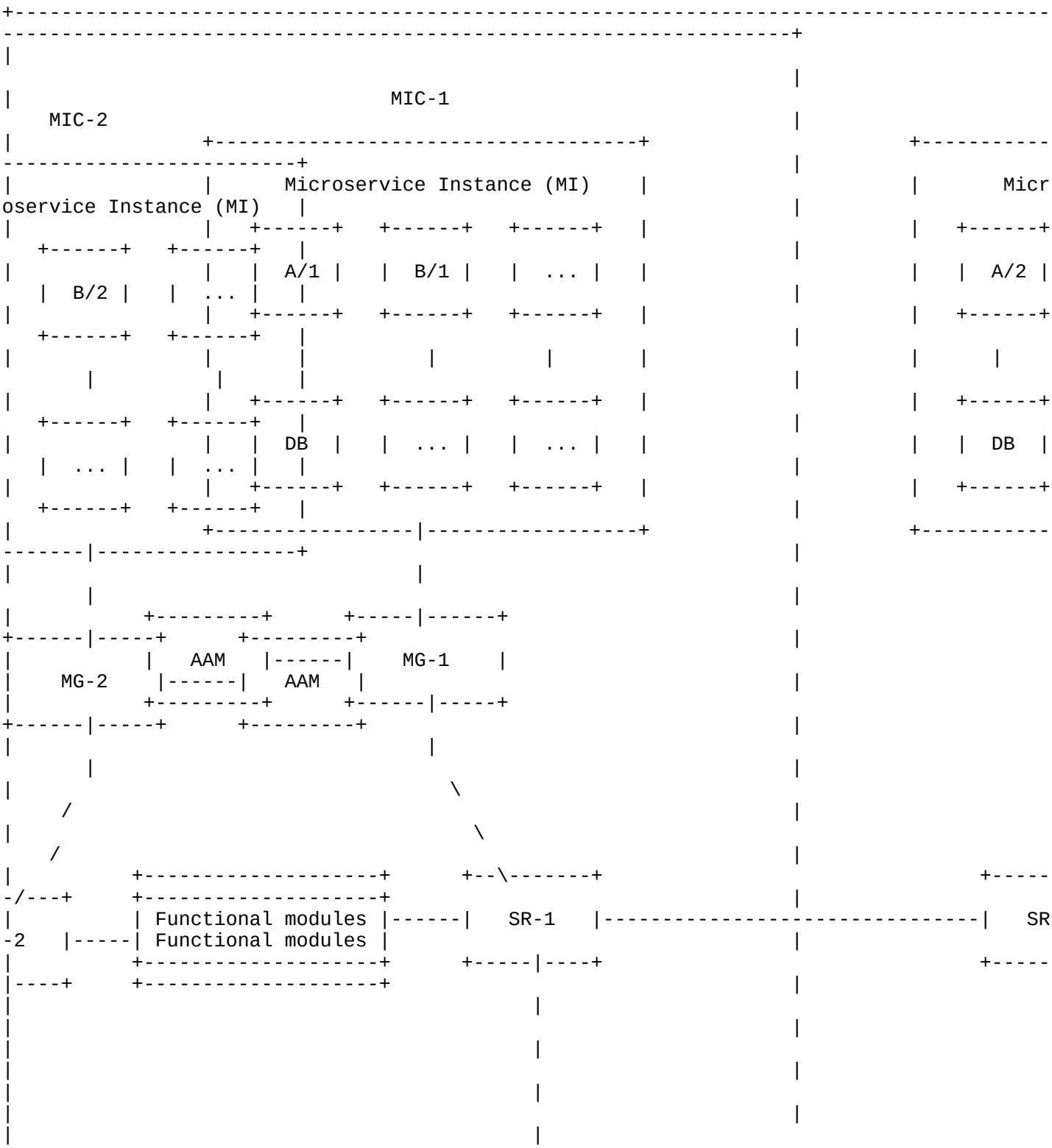
* SMO: Service Mesh Orchestrator, it is a key component in the microservice architecture responsible for coordinating, managing and optimizing the communication and task scheduling among microservices. It ensures the efficient collaborative work among services and dynamically adjusts the allocation of service traffic and computing tasks based on the actual load, performance requirements and system status.

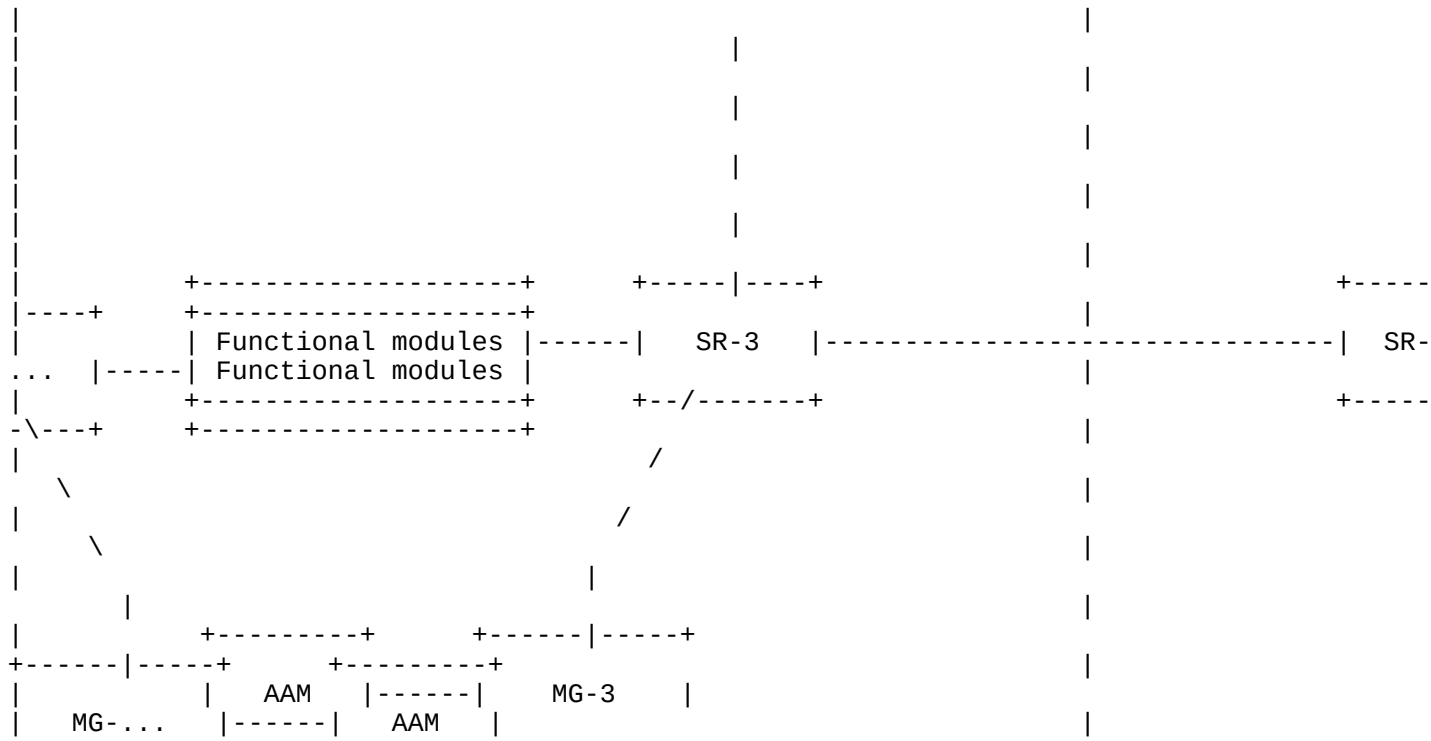
4. DMSC-LMT architecture

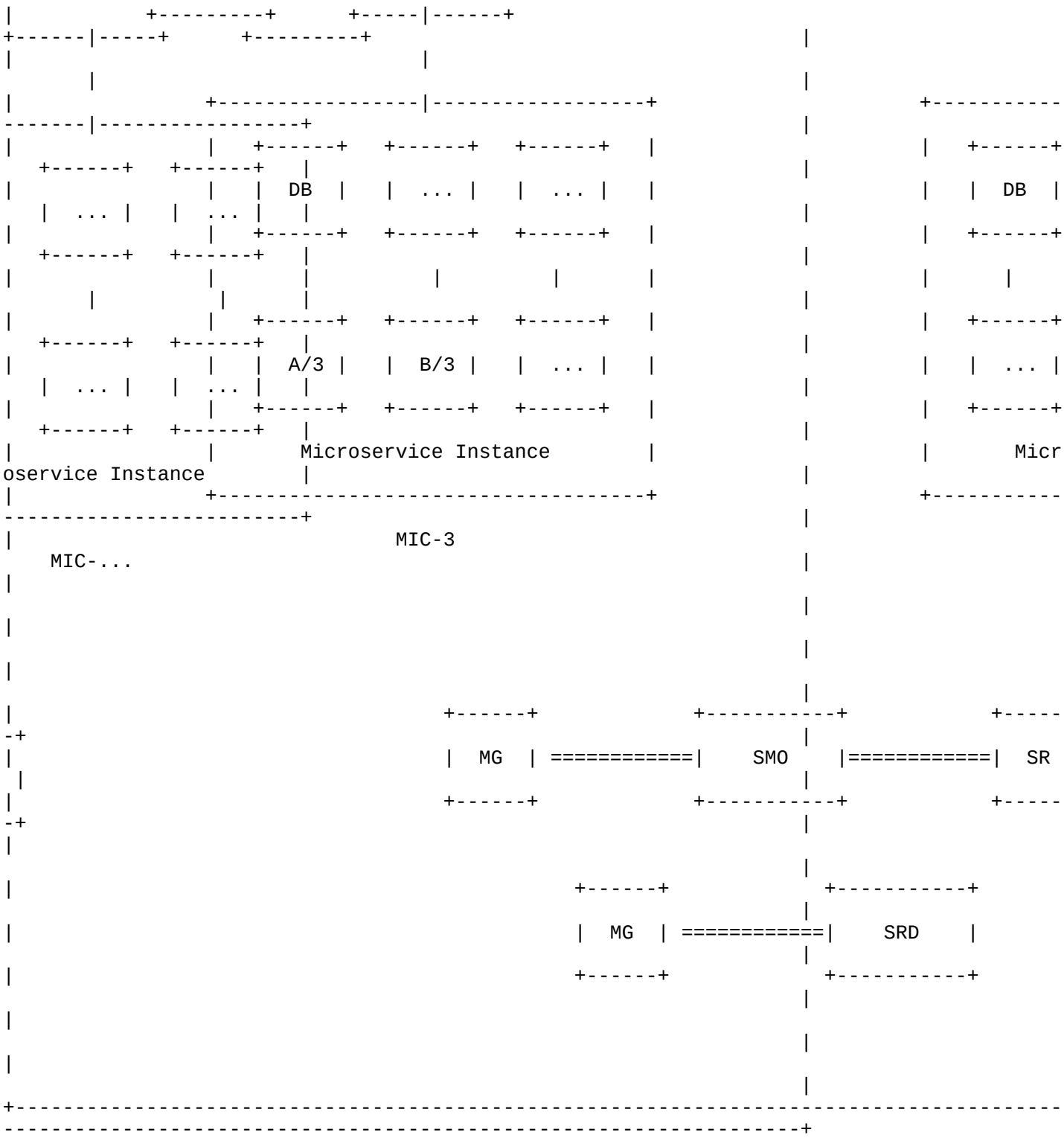
4.1. Overview

The Distributed Microservice Communication Architecture for Large Model Training (DMSC-LMT) is designed to support high-concurrency, high-throughput, and large-scale training scenarios. It decomposes the training system into modular microservice components to ensure flexible scheduling, efficient communication, and robust fault tolerance. The overall architecture of DMSC-LMT is shown in Figure 1. Microservice Instance Cluster (MIC) execute core training computations through parallelized microservices. Microservice Gateways (MG) serve as the unified traffic entry point and manage request routing. Authentication and Authorization Module (AAM) validate identity and enforce access control for all incoming

requests. Service Router (SR) coordinate internal service communication and load balancing. Service Registry and Discovery (SRD) manages microservice registration and discovery, enabling SR and MG to dynamically locate and route to service instances. Functional Modules provide reusable, loosely-coupled components— such as AI-driven congestion prediction and computation-aware routing— that may interact with external stateful services while remaining stateless in design.




+--------------------+
| Functional modules |
+--------------------+
SR-3
SR-...
MG-3
MG-...
AAM
AAM



4.2. Function modules

In DMSC-LMT, Functional Modules are loosely coupled components used to perform intelligent policy analysis, path optimization, and predictive reasoning. Each function module itself remains stateless, but it can access external state services, and has the characteristics of pluggable and reusable. To support flexible deployment and high-performance inference, Feature modules typically consist of the following core components as shown in Fig 2:

* Data Ingestion Interface: The functional module first obtains the input information required by the system during operation through the data acquisition interface. This includes data from MG and SR, module dependency topology from MIC, and service instance state from SRD. This data may be entered in many forms, such as RPC interfaces, REST apis, etc. This interface ensures that the module has full awareness of the current state of the system.

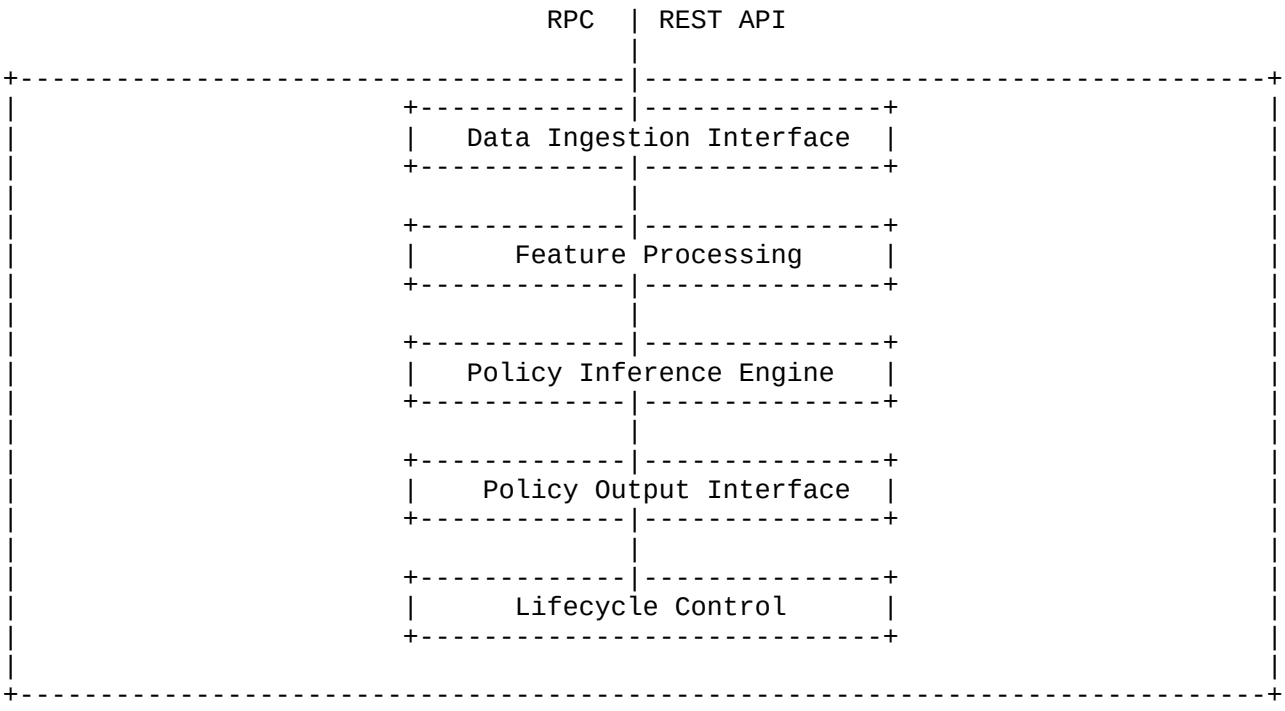
* Feature Processing: In order to adapt the data to the input requirements of the policy reasoning module, the feature processing and modeling components are integrated in the function module. This

part is responsible for preprocessing the original data, such as cleaning, normalization, sliding aggregation and so on. At the same time, it encodes the dependency topology between modules (such as generating adjacency matrix, graph embedding, etc.), and constructs the data structure suitable for graph model or sequence model analysis. Good feature modeling directly affects inference accuracy and performance.

* Policy Inference Engine: This functional module integrates a series of intelligent algorithms for decision generation. Different reasoning logics can be deployed in different modules, such as topology scoring based on graph neural network, or selecting the optimal routing path based on reinforcement learning strategy. Lightweight supervised learning models can also be used to classify and predict link congestion risk. This module receives the processed feature input and outputs policy suggestions, such as path priority, node avoidance suggestions, or traffic control parameters.

* Policy Output Interface: The inference results are fed back to the system control layer, mainly SMO, through this module. The interface supports synchronous or asynchronous transmission mode, and the generated policy recommendation is returned to the policy orchestrator in a standard format, which is adopted by the SMO and sent to the communication component (MG/SR). In addition, the output interface can also support logging, result echo or policy version control to enhance the traceability and interpretability of policy reasoning.

* Lifecycle Control: In order to ensure the independent deployment and hot update ability of the module, the life cycle control logic is included in the function module. This part is responsible for the management of module registration, starting, closing and version switching, and supports the dynamic enabling or disabling of functional modules according to requirements in different training tasks. Through this mechanism, the system can flexibly load different policy models according to the operation scenario without interrupting the core training process.



4.3. The control signaling messages design of DMSC-LMT

In this draft, we define the control signaling mechanisms of the Distributed Microservice Communication Architecture for Large Model Training (DMSC-LMT). This architecture introduces multiple communication entities and signaling types to support efficient task distribution, dynamic routing, security verification, and real-time system optimization in large-scale training environments.

DMSC-LMT introduces the following key communication entities:

* Service Registry and Discovery (SRD)

* Microservice Gateway (MG)

* Authentication and Authorization Module (AAM)

* Service Router (SR)

* Functional Modules

* Microservice Instance Cluster (MIC)

The control signaling messages exchanged among these entities are outlined in Figure 4. The signaling types and their respective functions are described as follows:

Service Instance Registration - Type 1: This signaling is initiated by microservice instances within MICs upon startup. Each instance registers its metadata (e.g., service prefix, resource type, compute capability,) with the SRD. This ensures that MG and SR can discover and route to active instances dynamically.

Service Route Advertisement - Type 2: This signaling is exchanged between SRD and SR. It distributes route availability and service prefix topology across the system. The SRD periodically broadcasts the status of all reachable service instances and their location to connected SRs, enabling accurate route calculation and service reachability.

Service Access Authentication - Type 3: When an external or internal request reaches MG, the MG sends an authentication signaling to AAM to validate the identity and permissions of the request. Based on policy rules, AAM returns an access decision, ensuring secure access control for all service invocations.

Compute-Aware Routing Notification - Type 4: This signaling is used by SRs to report current load conditions, hardware heterogeneity, and latency statistics of target service instances to MGs. Based on this feedback, MGs and SRs collaboratively determine the most efficient path for routing compute-intensive tasks.

QoS Telemetry and Policy Update - Type 5:Microservice Gateways (MGs) and Service Routers (SRs) periodically report communication quality metrics (e.g., throughput, error rate, latency) to the Service Mesh Orchestrator (SMO). The SMO then updates Quality-of-Service (QoS) policies and informs relevant entities to dynamically adjust traffic distribution strategies, ensuring optimal performance across the system.

4.4. DMSC-LMT communication flow

This section illustrates the communication flow among key components in the system during training task execution. It highlights how messages, control signals, and data are exchanged across Microservice Gateways (MGs), Service Routers (SRs), Service Mesh Orchestrator (SMO), and other entities to ensure coordinated training and resource-efficient operation.

1)Service registration and communication path configuration

After the training task is initiated, MG and SR first register their own node information, computing resources and service capabilities with SMO. The SRD synchronously completes the allocation of routing identifiers and maintains the logical mapping relationship of microservices. SMO combines the module topology information provided by MIC to determine the initial configuration scheme of the communication path and synchronously issues the strategy to each communication node.

2)Data exchange during the model training process

After the training task enters the execution stage, MG and SR conduct actual data transmission according to the configured communication strategy. MG is mainly responsible for the data entry and aggregation within the module, while SR is responsible for the forwarding of intermediate results and path selection between nodes. During the communication process, Functional Modules can participate in traffic prediction to avoid congestion and real-time optimization decisions.

3)Telemetry of communication quality and policy feedback

MG and SR continuously report communication indicators during operation, such as bandwidth occupancy, link delay, packet loss rate, etc. These telemetry information are aggregated and fed back to the SMO. Meanwhile, Functional Modules are also involved in the parsing of telemetry data to evaluate the current performance status of the system and predict its trends.

4)QoS policy update and traffic adjustment

Based on the observed data and the analysis results of functional modules, SMO will conduct a dynamic evaluation of the current QoS strategy. When problems such as path performance degradation, sudden increase in computing delay, or node load imbalance occur, the SMO will update the communication policy and distribute it to each MG and SR for execution.

5)Exception handling and fault recovery

During the communication process, once a node or link fails, MG or SR will actively report the abnormal state to SMO. SMO immediately links SRD and MIC for fault location and path reconstruction. Meanwhile, the functional module can conduct real-time analysis of events, such as determining whether it is an instantaneous fluctuation or a structural failure, thereby guiding the system on whether to retry in the short term, migrate tasks, or switch to a backup link.

6)Communication termination and resource cleanup

When the training task is completed, MG and SR will report the status to SMO and SRD. SMO is responsible for cleaning up communication routing information and deactivating service instances. SRD updates the service state mapping and releases port and routing resources. Meanwhile, MIC removes the module dependency topology of the current task.

4.5. Task-Level Microservice Decomposition

This section defines the implementation mechanisms of task-level microservice decomposition in DMSC-LMT. The decomposition process translates a large model training task into a set of loosely coupled microservice units, each corresponding to a specific computational stage, model layer, or data-parallel shard. These microservices are scheduled and deployed to Microservice Instance Clusters (MICs), registered with the SRD, and coordinated by the SMO for communication and resource optimization.

1)Decomposition Principles

Task-level microservice decomposition follows the principle of decoupling monolithic training workflows into independent computational sub-tasks with well-defined input and output boundaries. Each decomposed unit encapsulates a specific stage of the training pipeline, such as data loading, feature extraction, forward pass, backward propagation, or optimization. These units are designed to operate independently with minimal state sharing, thereby enabling stateless execution where possible. Furthermore, they are schedulable on heterogeneous hardware platforms, allowing flexible mapping based on compute capabilities or affinity constraints. The granularity of decomposition is determined by analyzing the model structure, the parallelism strategy employed—such as data, model, pipeline, or hybrid parallelism—and the trade-offs between communication overhead and task isolation. The overarching goal is to strike an effective balance between atomic execution units and manageable coordination complexity.

2)Service Unit Generation and Mapping

Once the decomposition strategy is determined, each sub-task is instantiated as a microservice unit, assigned a unique service identifier (SID), and bundled with its corresponding execution context. These microservice units are described through metadata, which includes information such as the SID, input/output schema, compute requirements, and dependency tags. Upon creation, each unit is registered with the SRD to facilitate dynamic service discovery.

The units are then mapped onto a logical task graph, where the edges represent communication dependencies between services. This mapping process may incorporate various constraints, including affinity requirements (such as GPU placement), execution order enforcement (e.g., layer-wise propagation), and co-location preferences to minimize cross-node latency. The resulting task graph abstraction is consumed by the SMO, which uses it to generate an initial deployment and routing plan for efficient execution across the system.

3)Deployment to MICs

Microservice instances are deployed to one or more MICs based on available resources, physical proximity, and the overall communication layout. The deployment process involves allocating container or runtime slots on MIC nodes, establishing routing configurations between dependent service instances via the SR and SRD, and registering each instance' s runtime status— such as active, pending, or faulted— to support system-wide monitoring and orchestration. The deployment mechanism is designed to be elastic, allowing new instances to be dynamically spawned or migrated across MICs in response to load fluctuations, node failures, or strategy updates issued by the SMO.

4)Dynamic Recomposition and Rescheduling

To handle runtime dynamics such as failures, congestion, or workload imbalance, DMSC-LMT supports dynamic recomposition and task rescheduling. Recomposition allows microservice instances to be regrouped or restructured, altering the communication topology to improve system performance or enhance fault isolation. Rescheduling, on the other hand, involves reallocating microservice instances to different MICs based on updated resource availability, QoS constraints, or degradation detected along communication paths. These adjustments are triggered by feedback from Functional Modules— such as congestion prediction or node pressure estimation— policy updates from the SMO based on runtime metrics and operational reports from SR and MG, as well as failure detectors within MICs that identify stalled or underperforming service chains. During the recomposition process, the SRD updates routing identifiers and service reachability maps, while the SMO coordinates configuration synchronization across affected nodes. This mechanism enables DMSC-LMT to maintain high throughput, adaptivity, and resilience under dynamic runtime conditions.

5. Realization of key functions

5.1. Efficient Microservice Communication and Orchestration

The system orchestrates and dynamically adjusts the communication strategy through SMO. SMO generates the routing strategy based on the task topology, system status, and the real-time analysis results provided by Functional Modules. The SRD is responsible for the registration and discovery of microservices, ensuring that MG and SR can dynamically locate communication targets and establish available paths. The functional modules play an intelligent auxiliary role in it. For example, through means such as congestion prediction, load awareness and path optimization, the adaptability and efficiency of routing decisions are improved.

5.2. Intelligent decision support for functional modules

To enhance the adaptive ability of the communication strategy, the system integrates a set of loosely coupled Functional Modules, such as AI-driven congestion predictors, computationally aware routers, etc. These modules access the telemetry data stream (from MG/SR) during the system operation and output policy suggestions for use by the SMO. Although functional modules may rely on external stateful services (such as historical data analysis), they are designed to be reusable, pluggable, and have good scalability and service independence.

5.3. Task topology and dependency control

MIC is responsible for analyzing the execution dependencies among the model modules and modeling them as an interaction topology diagram. During the task initiation stage, this topology is passed to the SMO as the basis for communication path design and routing strategy generation. Each microservice (such as MG and SR) registers its own instance information with the SRD at startup. The SRD is responsible for centrally maintaining the accessibility information of these services to support dynamic discovery and location during subsequent communications. MIC also participates in exception judgment during operation (such as identifying whether communication failures have disrupted the dependencies between modules), assisting the system in fault handling and module-level resscheduling.

5.4. Computationally aware routing and congestion avoidance mechanisms

To cope with the high concurrent communication and dynamic changes in computing power load in large-scale training tasks, DMSC-LMT introduces a set of computationally aware routing and AI-driven congestion avoidance mechanisms based on functional modules. This mechanism combines node computing pressure, communication link status and topological structure information to achieve dynamic perception,

prediction and optimal scheduling of communication paths between services.

A series of artificial intelligence learning algorithms have been deployed in the functional modules for predictive modeling and decision optimization of communication states. For example, topology-aware modeling based on Graph Neural Network (GNN); Routing optimization strategy based on reinforcement learning (RL); Path classification model based on supervised learning; Link trend modeling based on time series prediction.

These policies operate in a stateless, loosely coupled manner, analyzing the system state in real time and providing policy recommendations to the SMO. Based on this, SMO dynamically generates and issues communication strategies to realize real-time adaptive adjustment of service routes, avoid congestion paths, and improve the overall training efficiency and communication stability.

5.5. Fault handling and communication resilience

The DMSC-LMT system constructs a multi-level fault handling mechanism in the communication path and module interaction process, which is used to deal with node failure, link interruption and performance degradation, so as to ensure the continuity and stability of the training task.

The system monitors the status of services and links in real time through the exception reporting mechanism of MG and SR. Once an anomaly is detected, the information is immediately reported to the SMO. SMO combined with MIC analyzes the impact range of the fault. If the impact is limited, the local recovery strategy is preferred, such as switching the standby instance or adjusting the communication path, to avoid interrupting the task execution.

6. Local Database Integration and State Management in DMSC-LMT

In DMSC-LMT, each microservice instance is optionally equipped with a lightweight local database to support task-specific state persistence, intermediate result caching, runtime metadata storage, and execution traceability. While this decentralized storage model aligns with the architecture's goals of modularity, scalability, and low-latency operation, it introduces several technical considerations in terms of data consistency, recovery, and coordination.

6.1. Roles and Functions of Local Databases

Local databases within microservice instances serve the following core purposes:

*State Persistence: Retain intermediate training results, feature maps, local model slices, or cached gradients to avoid redundant computation and support resumption.

*Configuration and Dependency Storage: Hold task topology metadata, execution context, and module-specific parameters.

*Resilience and Fault Recovery: Enable fast recovery using local snapshots during instance restarts, crashes, or task migrations.

*Operational Metrics and Event Recording: Record policy inference decisions, communication statistics, fault signals, and historical performance metrics.

*Heterogeneous Execution Support: Cache localized execution parameters that vary by hardware or role.

*Version Control and Auditing: Track data versions, strategy revisions, and training progress checkpoints.

6.2. Coordination Across Instances

Although each microservice instance maintains its own local database, coordination is still necessary when tasks span across service boundaries. In DMSC-LMT, such coordination is achieved through topology-driven synchronization. The dependency graph of the decomposed training task is reflected in the service routing layer, where the SMO and SRD maintain a global view of service relationships and control data exchange accordingly. For example, when one microservice instance produces intermediate results required by a downstream module, the communication layer ensures timely delivery based on this task graph, without requiring direct access to each other's databases. This strategy simplifies implementation, minimizes coupling, and ensures consistency by enforcing dependency-aware data movement. It also allows SMO to monitor inter-instance flows and adjust paths or schedules when necessary, further enhancing the robustness of coordinated execution.

6.3. Integration with Control Plane Components

In the DMSC-LMT architecture, local databases are not isolated components but are designed to work in concert with the control plane. They serve as a critical bridge between the execution layer of training tasks and the decision-making logic of system orchestration. The SMO can instruct a microservice instance to create and save a persistent checkpoint, or to export a portion of its runtime state for use in global scheduling. A persistent checkpoint refers to a snapshot of the instance' s key training

state— such as model parameters, optimizer states, or execution progress— stored in the local database or stable storage. This enables the system to resume tasks from the last saved point in the event of migration or failure, avoiding redundant computation and ensuring continuity. Additionally, Functional Modules may access structured state data from the local store to support analytical tasks such as anomaly detection, retry path inference, or policy validation. These interactions typically occur through lightweight, read-only APIs that expose only necessary information. This approach preserves microservice autonomy and security while enhancing system observability and cross-module coordination. Through this integration, the local database not only handles internal state management but also plays an active role in the broader control logic of the system, supporting the adaptive, resilient, and intelligent orchestration goals of DMSC-LMT.

7. The specific process of DMSC-LTM

DMSC-LTM aims to achieve efficient and flexible distributed training by means of microservices. The architecture decomposes the training task of large-scale model into multiple microservice units, and uses the dynamic communication and scheduling between microservices to achieve efficient task allocation, load balancing and fault tolerance.

7.1. Overview of Large Model Training Parallel Strategies

With the increasing complexity of large models, the computing power of a single node can no longer meet their training requirements. Therefore, the parallel training strategy becomes the key to solve this challenge. Different parallel strategies offer solutions based on the model characteristics and training task requirements. Common large model training parallel strategies include Data Parallelism (DP), Pipeline Parallelism (PP), Tensor Parallelism (TP), and Hybrid Parallelism. These parallel strategies can be used independently or in combination to meet more complex training needs.

1)Data Parallelism (DP)

Data parallelism is one of the most commonly used parallel training methods. The basic idea is to divide the training dataset into multiple subsets, with each node processing a subset of the data. Each node computes gradients locally and synchronizes the gradients at the end of each training step to update the model parameters. Data parallelism is suitable for large-scale datasets, but when the model is large, the computation and storage capacity of a single node may become a bottleneck.

2)Pipeline Parallelism(PP)

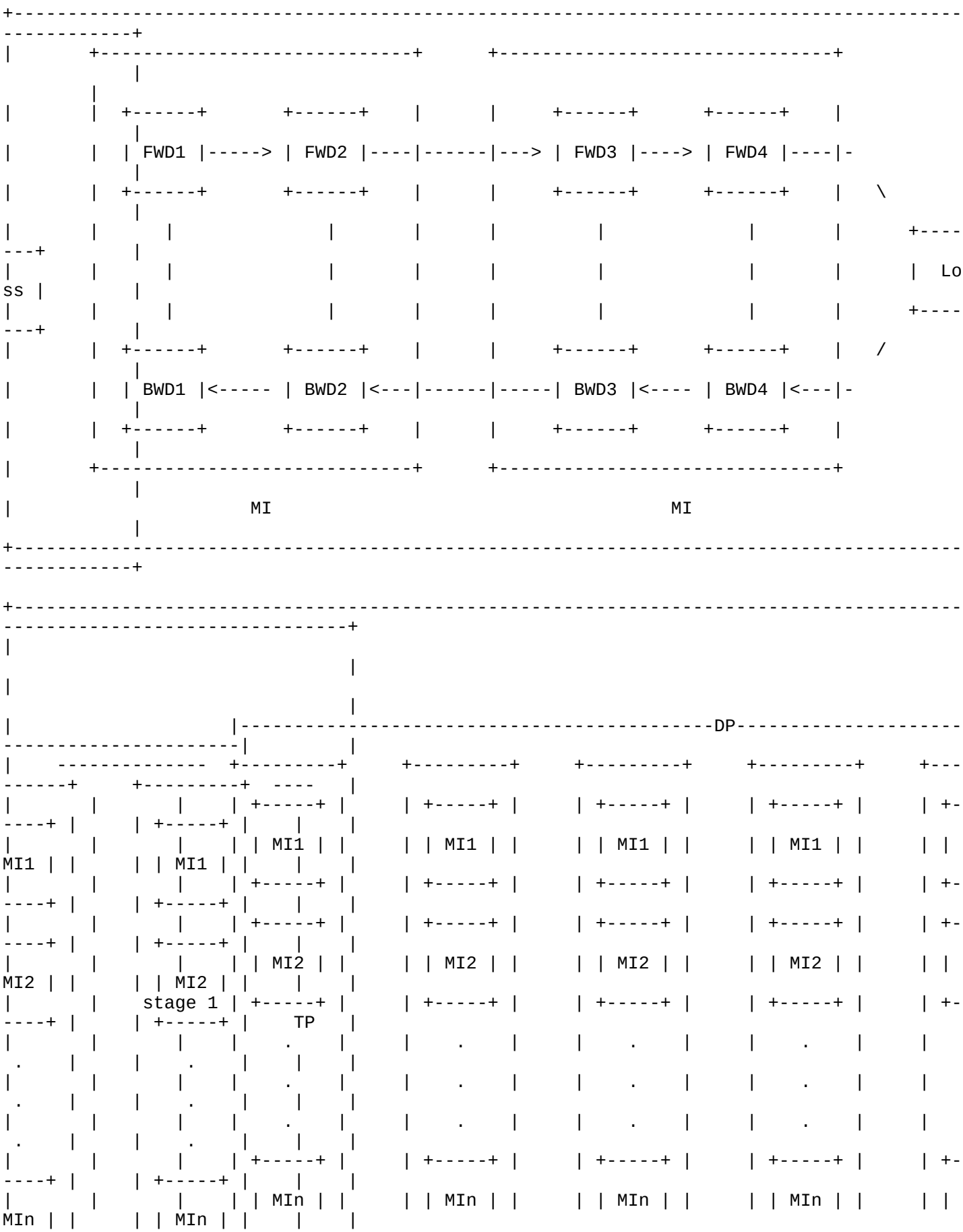
Pipeline parallelism splits a large model into several stages, with each stage assigned to a different computation node. Each node processes its assigned part of the model sequentially, and as data flows through the pipeline, different stages of the model are computed in parallel. This method is particularly useful for large models where each stage of the model requires substantial computation, allowing the model to be processed in a pipeline fashion, with each node focusing on a specific layer or subset of the model. Pipeline parallelism allows for better utilization of computational resources and reduces idle time between stages.

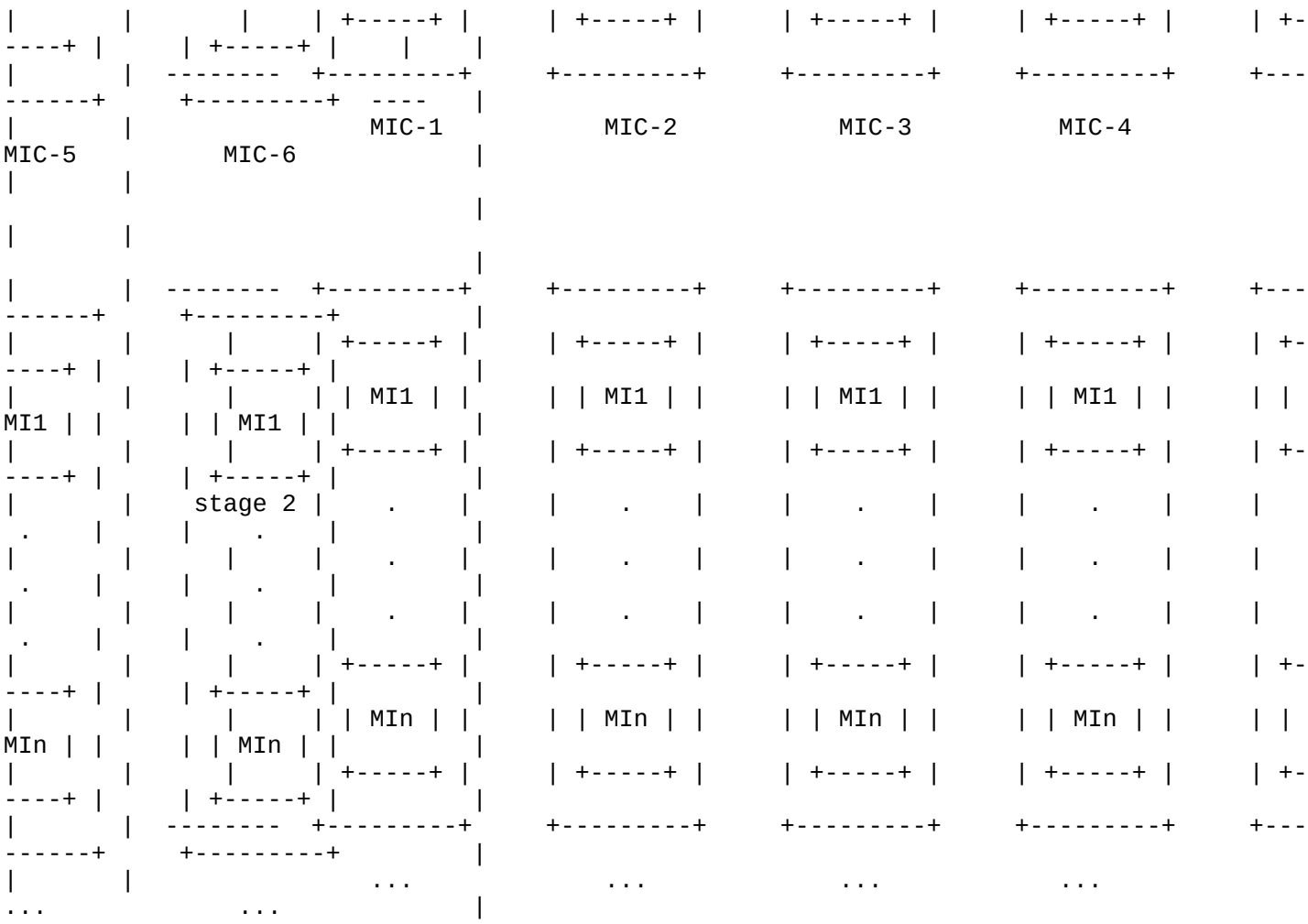
3)Tensor Parallelism (TP)

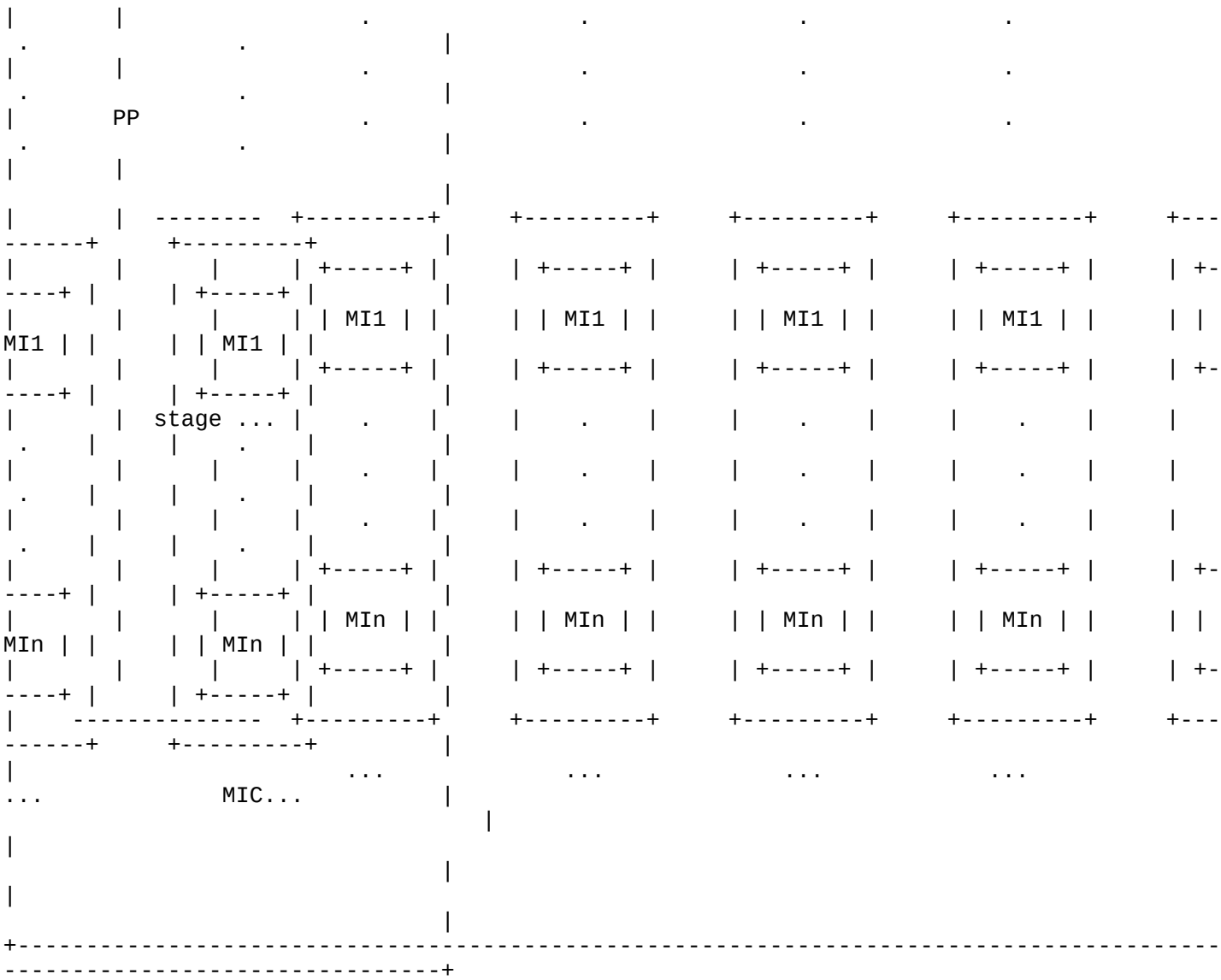
Tensor parallelism divides the tensors (e.g., weight matrices, gradients) in the model into smaller parts, with each node computing operations on a portion of the tensor. The advantage of tensor parallelism is that it effectively handles high-dimensional tensors, avoiding memory bottlenecks when a single node computes large, high-dimensional data. It is suitable for large-scale neural networks where parameter size and computational complexity are significant.

4) Hybrid Parallelism

Hybrid parallelism combines the advantages of data parallelism, pipeline parallelism, and tensor parallelism, dynamically selecting the most appropriate parallel strategy based on task requirements. In some complex large-scale model training scenarios, multiple parallel strategies may need to be used simultaneously to optimize the use of computational resources. For instance, data parallelism can be combined with pipeline parallelism to handle both data partitioning and model splitting, thus improving training efficiency. Hybrid parallelism is particularly useful for complex tasks that involve processing both large datasets and massive models.


FWD1
FWD2
FWD3
FWD4
Loss
BWD1
BWD2
BWD3
BWD4
MI
MI
DP
stage 1
TP
MI1
MI2
MIn


MIC-1
MIC-2
MIC-3
MIC-4
MIC-5
MIC-6
stage 2
MI1
MIn



7.2. Detailed mapping of distributed training to DMSC-LTM

Distributed training for large models involves various stages such as data preprocessing, model partitioning, forward and backward propagation, gradient synchronization, error recovery, and training completion. These stages are efficiently handled by DMSC-LTM through its microservices-based architecture and dynamic communication, routing, and resource scheduling capabilities. This section details how the components of DMSC-LTM map to each stage of the distributed training process, with Pipeline Parallelism (PP) used as an example.

1)Data Preprocessing and Loading

The first step in the distributed training process is data preprocessing, which typically involves normalization, augmentation, and feature extraction. In the DMSC-LTM architecture, preprocessing tasks are distributed across multiple MIs, with each instance responsible for a portion of the data. The MG acts as the entry point, receiving incoming data and routing it to the appropriate MI for processing. The SRD mechanism ensures that new preprocessing instances are dynamically registered and available for task assignment, providing flexibility and scalability during data

processing.

2)Model Partitioning

Once the data is ready, the model needs to be partitioned into smaller, manageable components, which can then be distributed across different computing nodes. In DMSC-LTM, each MI is assigned a specific part of the model to process. The SR mechanism ensures efficient communication between microservices. Here, functional modules—such as AI-driven congestion prediction and computation-aware routing—are employed to optimize the routing of data and tasks. These modules predict potential bottlenecks and dynamically adjust routing paths to avoid delays, ensuring the model is partitioned and executed efficiently. The SMO ensures that the model partitioning is done optimally, based on current system conditions, balancing the workload and minimizing resource wastage.

3)Forward Propagation

In DMSC-LTM, each MI is responsible for executing forward propagation for its specific part of the model. Using PP, data flows sequentially through each stage of the model, with each MI computing its assigned segment in parallel with other stages. The output of one MI is routed to the next using SR, ensuring smooth data transfer and efficient computation. Intermediate results can be cached for reuse in backward propagation, minimizing redundant computations and optimizing training performance.

4) Backward Propagation

After the forward pass is completed, backward propagation is performed to compute the gradients needed to update the model parameters. In DMSC-LTM, each MI calculates the gradients for its assigned model portion and sends the gradients to the previous MI in the pipeline using SR. Gradient synchronization is critical in distributed training, and DMSC-LTM leverages functional modules to optimize the synchronization process, ensuring efficient communication and minimal delay. This enables smooth backpropagation across multiple microservices and helps maintain high training performance.

5)Gradient Update and Parameter Synchronization

Following backward propagation, each MI updates its portion of the model using the computed gradients. To maintain consistency across the distributed system, parameter synchronization is required. In DMSC-LTM, SR are used to ensure that model parameters are updated across all microservices, guaranteeing that the model remains synchronized. The MG ensures that the final updated parameters are accessible to external systems for evaluation, deployment, or further training.

6)Error Handling and Recovery

Distributed systems are prone to errors such as node failures, network issues, or computation errors. In DMSC-LTM, the SMO is responsible for managing error handling and recovery. If a failure occurs, the SMO dynamically reallocates tasks to available microservices to minimize disruption. It also adjusts the task allocation based on system performance and load. Additionally, functional modules monitor the health of the system and apply AI-driven predictions to detect failures before they occur, enabling proactive recovery actions. This dynamic error recovery ensures that the distributed training process continues without significant interruptions.

8. Conclusion and Outlook

The DMSC-LMT architecture supports the scalability and flexibility of large-scale model training through task-level microservices, content-aware communication, functional module inference, and local state management. Its decentralized scheduling mechanism and microservice autonomy eliminate the centralized bottlenecks typical of traditional architectures, enhancing both flexibility and resource utilization efficiency. The pluggable intelligent decision-making mechanism enables dynamic adjustments based on varying training demands, further improving adaptability. Looking ahead, DMSC-LMT will continue to evolve in areas such as cross-tenant isolation, enhanced security, and cross-cluster deployment optimization, aiming to improve resource management capabilities in multi-tenant environments, ensure the security of training data and models, and optimize the performance and efficiency of large-scale training across global deployments.

9. IANA Considerations

TBD

10. Acknowledgement

TBD

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[InfRef] "", 2004.

Appendix A. An Appendix

Authors' Addresses

Hui Yang
Beijing University of Posts and Telecommunications
10 Xitucheng Road, Haidian District
Beijing
Beijing, 100876
China
Email: yanghui@bupt.edu.cn

Tiankuo Yu
Beijing University of Posts and Telecommunications
10 Xitucheng Road, Haidian District
Beijing
Beijing, 100876
China
Email: yutiankuo@bupt.edu.cn

Qiuyan Yao
Beijing University of Posts and Telecommunications
10 Xitucheng Road, Haidian District
Beijing
Beijing, 100876
China
Email: yqy89716@bupt.edu.cn

Zepeng Zhang
Beijing University of Posts and Telecommunications
10 Xitucheng Road, Haidian District
Beijing
Beijing, 100876
China
Email: 2024140574@bupt.cn